MA 3046 - Matrix Analysis
Laboratory Number 11
Iterative Solution of Systems of Linear Equations

As we have already discussed numerous times, numerical linear algebra involves some fundamentally different considerations than apply to more general numerical approximations, e.g. the approximation of integrals, derivatives, etc. One primary reason for this is that, in theory,

$$\mathbf{A}\,\mathbf{x} \;=\; \mathbf{b}$$

is a problem which requires **no** approximations! Elementary linear algebra classes demonstrate that this problem is exactly solvable, using infinite precision arithmetic, by any of several variants of Gaussian elimination, and in a finite number of steps. (Moreover, especially efficient variants of such methods exist for solving commonly-occurring cases where $\mathbf{A}$ has some special *structure*, e.g. banded matrices.) Because they theoretically produce the exact solution, Gaussian elimination and its relatives, e.g.Gauss-Jordan elimination, $\mathbf{L}\,\mathbf{U}$ decomposition, etc., are commonly referred to a *Direct Methods*.

However, as we have also discussed, elementary linear algebra classes also generally fail to address exactly how large the magnitude of this (albeit finite) number of computations required by direct methods may actually be. In fact, for any of the Gaussian-elimination variants, this number can be shown to be roughly proportional to m^3, where m is the number of rows and columns in $\mathbf{A}$. (The proportionality becomes more exact the larger the value of m.) We find this oversight unfortunate because, "real" application matrices can be quite large, and therefore, in practice, the number of computations required when, say for example, $n \sim 10^5$, can become rather daunting. Consequently, the corresponding time required for solution may become quite unsatisfactory, even on what we would generally think of as relative fast computers. Moreover, for physical reasons, many "real" problems involve relatively *sparse* matrices $\mathbf{A}$, i.e. ones composed mainly of zeros. Unfortunately, methods based on Gaussian elimination frequently cannot exploit this structure, since the elementary row operations required by the elimination process tend to fill-in many of the locations originally occupied by zeros with non-zeros resulting from the required calculations.

Because of such considerations, a second, general class of methods, so-called *Iterative Methods* have been developed. These methods traditionally involve reformulating the original problem into a new, equivalent one of the form

$$\mathbf{x} = \mathbf{G}\mathbf{x} + \mathbf{b}' \tag{1}$$

for which algorithm:

$$\mathbf{x}^{(k+1)} = \mathbf{G}\mathbf{x}^{(k)} + \mathbf{b}' \tag{2}$$

converges. Such methods are especially attractive when $\mathbf{A}$ is sparse, because for many implementations, $\mathbf{G}$ will contain only about as many non-zero elements are were in the original matrix $\mathbf{A}$, and the multiplications in (2) need only be actually done for the non-zero

elements of $\mathbf{G}$. Different iterative methods, of which the most commonly-encountered are Jacobi, Gauss-Seidel, Successive Over-relaxation (SOR), and Conjugate Gradient methods, arise from different constructions of the iteration matrix $\mathbf{G}$. More modern methods involve so-called Krylov space iterations. While consideration of such methods is beyond the scope of this laboratory, we would note that their practical success still required a formulation where matrix multiplication can be done fairly cheaply!

The matrix $\mathbf{G}$ is also commonly called the *splitting* matrix, because for many methods, it can be constructed by splitting the matrix $\mathbf{A}$ in the original system into several parts, and then moving some of those parts from one side of the equality to the other. For example, the matrix $\mathbf{A}$ can be written:

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U} \ ,$$

where $\mathbf{D}$ consists of the diagonal elements of $\mathbf{A}$, $\mathbf{L}$ contains the elements from below the diagonal, and $\mathbf{U}$ the elements above. (Note these are **not** the same as the $\mathbf{L}$ and $\mathbf{U}$ from the Gaussian elimination $\mathbf{L}\,\mathbf{U}$ decomposition.) In terms of these matrices, the original system then becomes

$$(\mathbf{D} \ + \ \mathbf{L} \ + \ \mathbf{U}) \ \mathbf{x} \ = \ \mathbf{b}$$

or, equivalently

$$\mathbf{D}\,\mathbf{x} \ = \ -(\ \mathbf{L} \ + \ \mathbf{U}\ )\ \mathbf{x} \ + \ \mathbf{b}$$

or

$$\mathbf{x} \ = \ \underbrace{-\mathbf{D}^{-1} \ (\ \mathbf{L} \ + \ \mathbf{U}\ )}_{\mathbf{G}} \ \mathbf{x} \ + \ \underbrace{\mathbf{D}^{-1}\ \mathbf{b}}_{\mathbf{b}'} \ , \tag{3}$$

which is of the required form with $\mathbf{G}$ and $\mathbf{b}'$ as shown. We commonly call the iterative form of equation (3), i.e.

$$\mathbf{x}^{(k+1)} \ = \ -\mathbf{D}^{-1} \ (\ \mathbf{L} \ + \ \mathbf{U}\ )\ \mathbf{x}^{(k)} \ + \ \mathbf{D}^{-1}\ \mathbf{b} \ , \tag{4}$$

the Jacobi method. This method is implemented in the program **jacobi.m** as shown in Figure 11.1. (Note that this program is designed solely to show the convergence of the algorithm. In fact, as written, it exemplifies exactly how **NOT** to program the jacobi algorithm, since by using full matrices, backslashes, etc., it in effect discards exactly the feature, i.e. the need to work with only the non-zero elements, that make iterative methods attractive.) Note also this program makes use of the MATLAB functions **diag( )**, which can either extract or create the diagonal elements of a matrix, and the functions **tril( )** and **triu( )**, which extract, respectively, the upper and lower-triangular parts of a matrix (including the diagonal elements).

Other iterative methods which we shall also study in this laboratory are the Gauss-Seidel, method, which can be expressed as:

$$\mathbf{x}^{(k+1)} \ = \ \underbrace{-(\mathbf{D} \ + \ \mathbf{L})^{-1}\mathbf{U}}_{\mathbf{G}} \ \mathbf{x}^{(k)} \ + \ \underbrace{(\mathbf{D} \ + \ \mathbf{L})^{-1}\mathbf{b}}_{\mathbf{b}'} \ , \tag{5}$$

```
%
      clear
      format bank ;
      data = [ ]       ;
%
      for    N = [ 2 4 8 16 32 64 128 ] ;
%
            h = 1/N ;
%
            A(1,1:2) = [ 1 0 ] ;
            for k = 2:N
                  A(k,(k-1):(k+1)) = [ 1 -2 1 ] ;
            end
            A(N+1,N:(N+1)) = [ 0 1 ] ;
%
            L = A - triu(A) ;
            U = A - tril(A) ;
            D = diag( diag(A)) ;
            G = -D\(L+U)        ;
%
            b       = [ 0 ; -h^2*ones(N-1,1) ; 0 ] ;
            bprime =       D\b ;
%
            u = A\b ;
            conv = max( abs( eig(G))) ;
%
            xn       = zeros(N+1,1) ;
            niter = 0 ;
            while ( norm( u - xn ) > .000001    )
                  xn         = G*xn + bprime ;
                  niter = niter + 1 ;
            end
%
            data   = [ data ; N    h conv    niter ] ;
      end
%
      data ;
      format ;
```

Figure 11.1 - Listing of Program **jacobi.m**

and the SOR method:

$$\mathbf{x}^{(k+1)} = \underbrace{(\mathbf{D} + \omega\,\mathbf{L})^{-1}((1-\omega)\mathbf{D} - \omega\mathbf{U})}_{\mathbf{G}}\,\mathbf{x}^{(k)} + \underbrace{\omega(\mathbf{D} + \omega\,\mathbf{L})^{-1}\,\mathbf{b}}_{\mathbf{b}'} \qquad (6)$$

Interesting enough, iterative methods have another, small side benefit. This benefit accrues due to the fact that, in general, convergence of these methods is not dependent on the choice of $\mathbf{x}^{(0)}$, and therefore, round-off errors do **not** propagate. Put somewhat differently, each new iteration of (2) starts with an effectively "clean slate," and the fact that the $\mathbf{x}^{(k)}$ that is actually being used is slightly different from the "true" one simply means that iteration is starting with a slightly different initial guess. On the down side however, this fact does not change the more fundamental observation that the error encountered when solving a linear system satisfies the inequality

$$\frac{\|\,\mathbf{e}\,\|}{\|\,\mathbf{x}\,\|} \leq \kappa(\mathbf{A}) \cdot \frac{\|\,\mathbf{r}\,\|}{\|\,\mathbf{b}\,\|}$$

where $\kappa(A) \equiv \|\,\mathbf{A}\,\| \cdot \|\,\mathbf{A}^{-1}\,\|$ is the condition number of $\mathbf{A}$. Therefore, in general, applying an iterative method to a truly ill-conditioned problem makes no more sense than applying a direct method to the same problem.

Name: _____

MA 3046 - Matrix Analysis
Laboratory Number 11
Iterative Solution of Systems of Linear Equations

1. Copy to your local directory the file:

jacobi.m

and start MATLAB.

2. Give the MATLAB **help** command to review the **diag**, **tril** and **triu** functions. Once you have done this, use your texteditor to study the **m.** file **jacobi.m** until you are comfortable that it implements the algorithm given in (4).

Then run the program (you may need to be a bit patient, depending on the basic speed of your machine) and record the results:

3. Modify the MATLAB **.m** file **jacobi.m** so as to implement the Gauss-Seidel method (5). (You may want to copy **jacobi.m** into a new file, say **gauss_seidel.m** before making these modifications.

Then run the modified program. Does the observed behavior (compared to the results of the Jacobi algorithm) look reasonable and agree with the theory?

4. Lastly, modify the MATLAB **.m** file **jacobi.m** so as to implement the SOR method (6). (You may again want to make a new copy. Also, be very careful when computing **b'**. The MATLAB expression

$$\mathbf{w * (D + w * L) \setminus b}$$

does **not** produce what you might think it would!) Use for the value of the relaxation parameter (ω)

$$\omega = \frac{2}{1 + \sin\left(\frac{\pi h}{2}\right)}$$

Then run the modified program. Does the observed behavior (compared to the results of the Jacobi algorithm) look reasonable and agree with the theory?